DoorDash Challenge

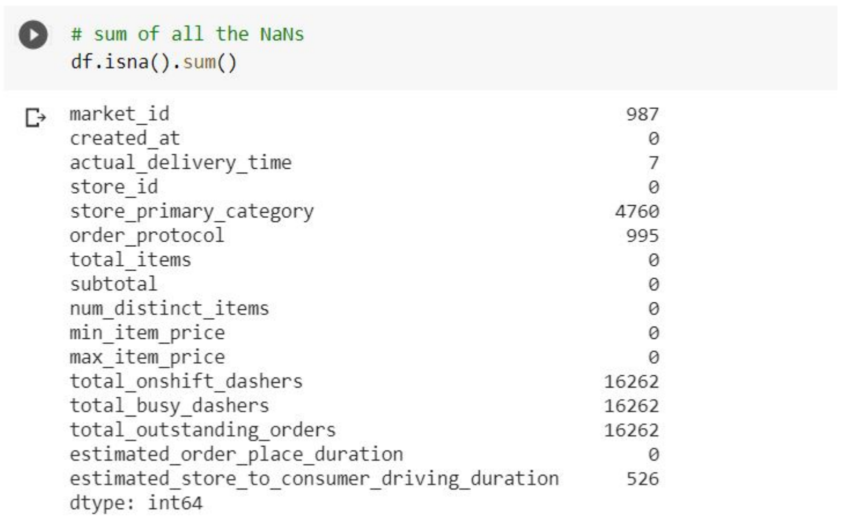
Prepared by: Chukwuemeka Okoli
Report Date: 9th March, 2021

Introduction

In this report, findings from the DoorDash project are presented. DoorDash is trying to predict the total delivery duration in seconds using historical data received in early 2015 in a subset of cities. The objective is to build a model to predict the total delivery duration seconds value. This is clearly a supervised learning task, since we are given labeled training data, and it is a typical regression task since the objective is to predict a value. More specifically, this is a multiple regression problem, since we are using multiple features to make a prediction. My main assumption was that there is no continuous flow of data coming into the system hence no need to adjust to changing data rapidly, so plain batch learning is fine for this project.

Data Preprocessing and Cleaning

I checked to see if my data had missing values and found that a few features had missing values as shown below. I



```
# sum of all the NaNs
df.isna().sum()
```

```
market_id                                          987
created_at                                           0
actual_delivery_time                                 7
store_id                                             0
store_primary_category                            4760
order_protocol                                     995
total_items                                          0
subtotal                                             0
num_distinct_items                                   0
min_item_price                                       0
max_item_price                                       0
total_onshift_dashers                            16262
total_busy_dashers                               16262
total_outstanding_orders                         16262
estimated_order_place_duration                       0
estimated_store_to_consumer_driving_duration       526
dtype: int64
```

Figure 1: Statistics of Missing Figures

created my response variable "*delivery_duration*" in seconds by converting "*created_at*" and "*actual_delivery_time*" to time data types and find the difference between these two features. I plotted scatter plots and histograms on all my attributes to check for outliers and I found that the feature - "*total_items*" have an outlier. The value was more than 3 times the standard deviation of the feature. I also checked features like "*num_distinct_items*" and "*subtotal*" to see if it correlates with that oddly high "*total_item*" but the values for distinct items was below 10 and the subtotal was relatively low, therefore, I went ahead and deleted that row from my dataset.

Also, from my visualizations, I noticed that some features had observations below zero which was odd. These features include "*min_item_price*", "*total_onshift_dashers*", and "*total_outstanding_orders*". They did not make sense for them to have observations below zero. I assumed this was an error in data collection and instead of deleting those instances which may cost my model crucial learning points, I decided to take the absolute values of these negative observations. I noticed some discrepancies with the "*min_item_price*" and "*max_item_price*" features. These discrepancies include:

- For "*total_items_order*" = 1, "*min_item_price*" and "*max_item_price*" observation was different from "*subtotal*"
- "*min_item_price*" was greater than "*max_item_price*"
- "*max_item_price*" was greater than "*subtotal*"

```python
# histogram of each numerical attribute
df.hist(bins=50, figsize=(25,20))
plt.show()
```

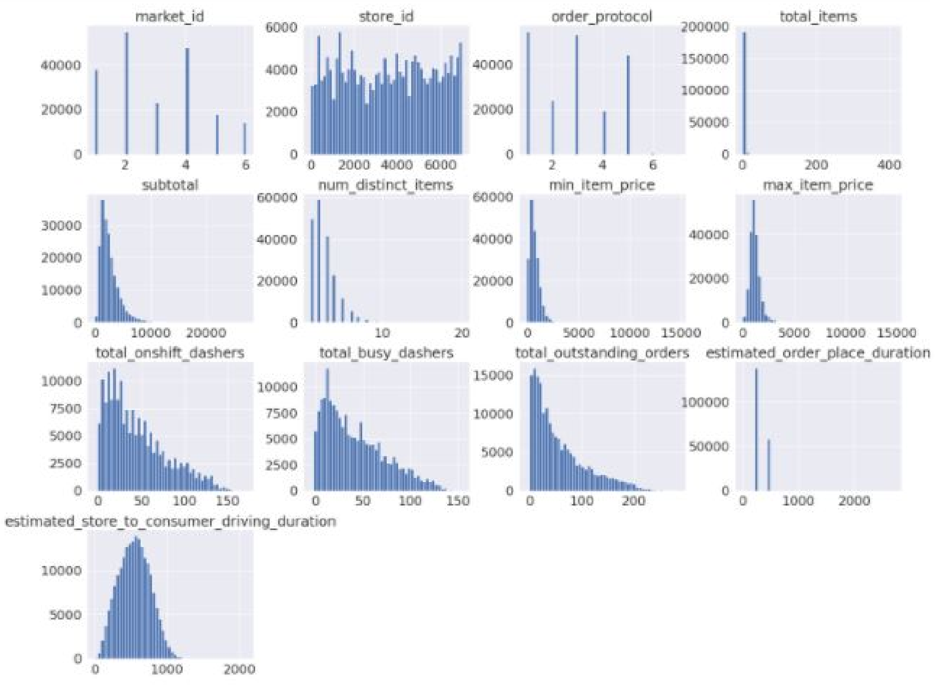


Figure 2: Histogram of features

I took care of these discrepancies in a few logical ways, as follows:
For descrepancy 1:

– when *total_items_order* = 1, I equated *min_item_price* and *max_item_price* to *subtotal*

For descrepancy 2:

– when *total_items_order* = 2 and *num_distinct_items* = 2 and *subtotal* - *min_item_price* < *min_price_item*, I made *max_item_price* equal to *min_item_price*, and *min_item_price* equal to the difference between *subtotal* and new *max_item_price*.
– when *total_items_order* = 2 and *num_distinct_items* = 2 and *subtotal* - *min_item_price* > *min_price_item*, I made *max_item_price* equal to the difference between *subtotal* and *min_item_price*.
– when *max_item_price* was greater than *subtotal*, I took the average of *max_item_price* and *min_item_price* and multiplied by *total_order* to get my *subtotal*.

I made these decisions instead of deleting the rows because they seemed logical and unbiased and preserved the data, thereby, providing my models more points for learning. I also noticed that the subset of *total_onshift_dashers*, *total_busy_dashers* was greater than *total_onshift_dashers* and took care of these discrepancies similar to the steps above.

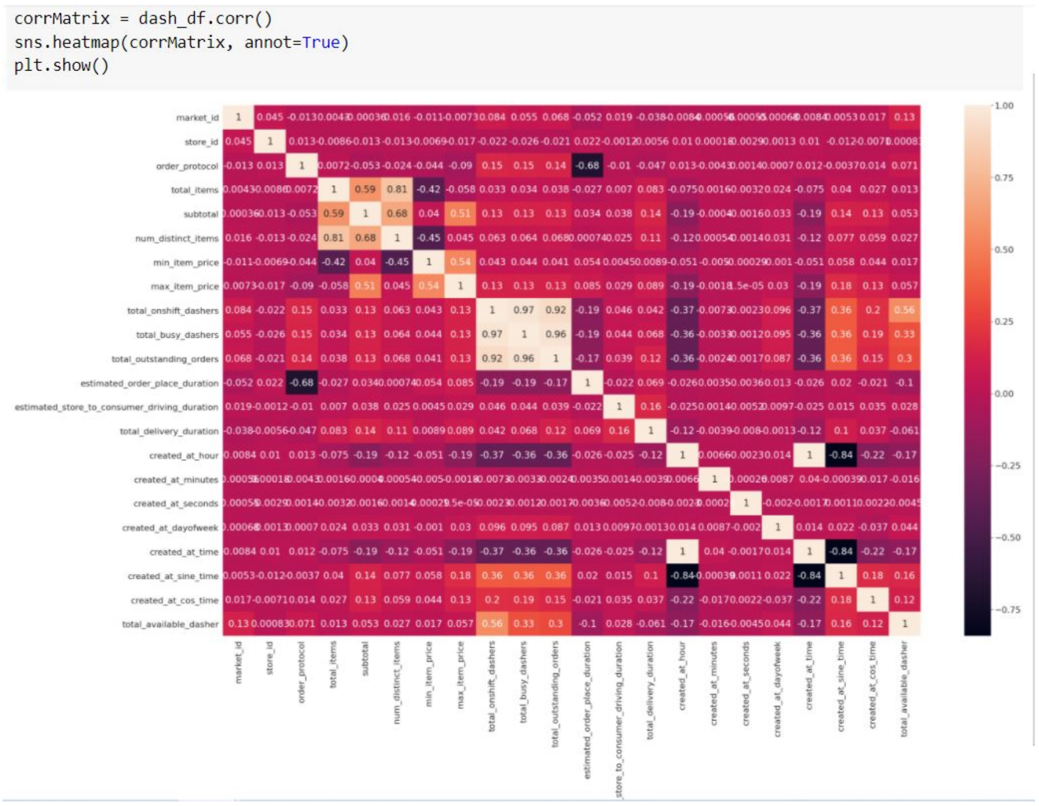
I noticed that three features - *total_onshift_dashers*, *total_busy_dashers* and *total_outstanding_orders* had 8% of their data mising. But before I decided to delete these observations I ran a correlation matrix to see how each predictor variable correlated with the response variable (*total_delivery_duration*). I noticed that these features had high correlation with my response variable and deleting these features was not the best act, therefore, I decided to delete the rows from my dataset that had this missing value (8% is not too much to lose).

Feature Selection and Engineering

I created a few new features from my datetime feature. I extracted the *day_of_the week* and the total time in seconds. I noticed there was only one year (2015) and two months (January and February) and decided to not add these feature as they will not help my model against new dataset that contained different months and years. I decided to include the day of the week feature and time feature because I believed it will have a big impact of delivery time estimation. This is because:

- During the week day, I assume there will be more orders as people don't have time to cook or eat out because of work and also traffic will be low during working hours except for afternoon lunch break.
- Also, the weekend will also affect the delivery time as there will be more traffic and probably less orders since people have more time to cook or eat at a restaurant.

Since time is a cyclic phenomenon and 2300hrs is closer to 0000hrs that 0500hrs is, I had to make use of cosine and sine periods of my time feature to let my model understand how time works. therefore, I created two new features of time in seconds, *created_at_sine_time* and *created_at_cos_time*. I created one more feature *total_available_dasher*, by subtracting the *total_busy_dashers* from *total_available_dasher*. I felt this will be a meaningful feature since it showed how many drivers are available for an order. I went ahead to create dummy variables for categorical features and features that were label encoded such as *market_id*, *store_primary_category*, *order_protocol*, *created_at_dayofweek*. I created dummies for non-categorical feature like *market_id* and so on because I did not want my model to derive, for example, that 4 is closer to 5 than it is to 1, which is entirely not true.



```
corrMatrix = dash_df.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```

Figure 3: Correlation matrix for features

Model Development, Evaluation and Refinement

I split my data set randomly in train and valid set (70:30 percent ratio) and went ahead to scale my predictor features. From **Figure 3** above it was clear that some predictors were more significant than the other and needed to be selected. Also, to reduce model overfitting due to bias, I decided to perform a principal component analysis (PCA) to determine the optimal subset size and features that best describe the response variable. I made use of the linear regression (least square fit) Forward Stepwise Selection technique for this purpose. Forward stepwise selection begins with a model containing no predictors and then adds predictors one at a time until all the predictors are in the model. I implemented this algorithm on my training dataset, and evaluated and validated the best subset based on its adjusted R^2, Mallow's C_p, Akaike's Information Criteria (AIC), and Bayesian Information Criteria (BIC). Algorithm for the forward stepwise selection is:

Let Mo denote the null model which contains no predictors

- For $k = 1, 2, ..., n-1$
- Consider all $n-k$ models that augment the predictors in M_k with one additional predictor
- Choose the best among these $n-K$ models, and call it M_{k+1}
- Select the single best model among M_0, M_1, ..., M_n using cross validated prediction error, C_p, BIC, adjusted R^2 or any other method.

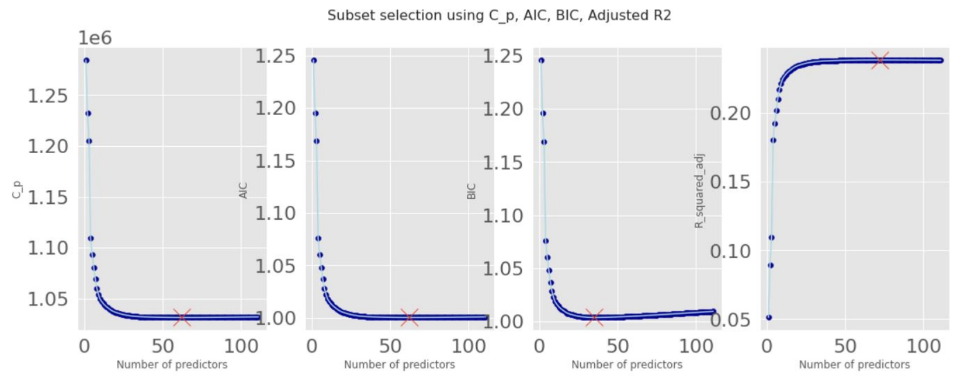


Figure 4: Plot of best feature subset by different evaluators

Figure 4 above shows the best feature subset selection from 111 predictor features. I decided to go with 65 subset of features since C_p and AIC evaluators had this same number of best feature subset. I moved forward to testing several models (Linear regression, Decision trees, Random Forest and XGBoost) to see how each fared with my dataset, and evaluated each model using cross-validation to detect overfitting. Below were the results.

Table 1: Results from test on several models

S/N	Model Algorithm	Mean RMSE (seconds)	Standard Deviation
1	Linear Regression	1002.9	54.46
2	Decision Tree	1401.66	47
3	Random Forest	973	57.9
4	XGBoost	952	58

From the table above, it is clear that XGBoost had the lowest RMSE and standard deviation error of a minute was acceptable. So, I decided to use XGBoost model.

Hyperparameter Tuning

I created a list of different values for XGBoost hyperparameter and ran a grid search using python's GridSearchCV package to determine the best selection of hyperparameters for my model. However, after waiting hours, it seemed like my computer was not fast enough and my computer ended up crashing so I decided to jump over this stage. However, with a better computer I would have been able to run my hyperparameter tuning.

Model Validation

I tested my XGBoost model against my valid subset and got a RMSE of 2347.25 secs and a Mean Average Percentage Error of 23.6%. This result seems decent to me.

Test Data

I read in my test data set and cleaned the data same way I did with my train data set. I also created identical features and used dummy variables for categorical features. I used my XGBoost model to make predictions and then converted my predictions (seconds) to *"Actual_delivery_time"* by adding those seconds to the *"created_at"* feature.

Additional Features needed

I would love to have had original dataset for "market_id", "order_protocol" as I felt this would have made it easier to create new features like traffic intensity (using three classes: very high, medium, low). In addition, nationality/race type of these cities will give more information into ordering habits which I can use to assign weights to different orders. "Order protocol" which I feel means the medium through which the order was placed (telephone call, mobile application and so on) would also be useful. I would also love to see data on how internet and network coverage affects delivery time.

Conclusion

The following features will be helpful in future analysis:

- Historical prep time per meal, current traffic patterns, restaurant's track record, restaurant's reviews online, delivery vehicle type, parking limitation at both restaurant and customer location.
- Restaurant's rating, issues with drop off, difficulty in accessing customer location.

Knowing historical prep time will be useful in prediction delivery time, current traffic pattern can be useful in route optimization, restaurant's online reviews and track record can give an indication of how fast restaurants responds to orders. The delivery vehicle type whether a bike or car can determine how fast a dasher picks up an order at the restaurant. The parking limitation at different restaurants and customer location should be factored in because an inaccessible apartment or restaurant will lead to an incomplete pickup/delivery. All these features when used correctly can be useful in predictive analytics and the decision making process with regards to customer's orders.

Having the right features and including good features will benefit any Machine Learning algorithm. Whether it is historical features like the average prep time in the last one week, near real-time features such as average prep time in the last few minutes, or real time features such as time of the day, hour of the day, order size etc. The ability to accurately predict delivery times is important to customer satisfaction and retention. Predicting actual delivery duration is very important both to the customer and the delivery partner. Time prediction is vital to dispatch delivery drivers (dashers) as time to dispatch driver can affect real-time food delivery among other factors.

A very good model is as good as the data it see and this is why data collection, warehousing, extraction, transformation and loading processes are vital too. Data cleaning and wrangling is important in the Machine Learning pipeline. I believe my performance will match up to a model already in production. Every feature is an important metric as a combination of features can ensure efficient real-time prediction and analytics. Machine Learning is a tool that has been proven to be effective in taking all three-sided customers (deliver drivers, restaurants, and consumers) in the marketplace into account during decision making and analytics.

On the business side of things, one way to boost business and improve customer service is to develop ways of enticing drivers to go online or pickup an order. One of such ways could be:

- Introducing air miles on eligible trips for drivers for instance 100 air miles point for every 1000 miles dash.
- Route optimization to reduce total delivery time.

At the end of this project, I was able to build a model to predict the total duration delivery seconds using Machine Learning by generating additional features. This model can be fine-tuned and streamlined into the existing process to aid decision making in business.

Next Step

As a next step, I would like to see how Machine Learning can be used to:

- Predict which driver is likely to drop an order once he/she have accepted the order and/or arrived at pick-up point.
- Predict the likelihood of a driver to accept and complete an order